CIC Compiler v4.0

LogiCORE IP Product Guide

Vivado Design Suite

PG140 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

IP Facts

Chapter 1: Overview

Licensing and Ordering Information 5

Chapter 2: Product Specification

Port Descriptions 6

Chapter 3: Designing with the Core

General Design Guidelines 9
Clocking 19
Resets 19
Protocol Description 19

Chapter 4: Design Flow Steps

Customizing and Generating the Core 26
Constraining the Core 33
Simulation 33
Synthesis and Implementation 34

Chapter 5: Test Bench

Demonstration Test Bench 35

Appendix A: Migrating and Upgrading

Migrating to the Vivado Design Suite 37
Upgrading in the Vivado Design Suite 37

Appendix B: Debugging

Finding Help on Xilinx.com 46
Interface Debug 47

Appendix C: Additional Resources and Legal Notices

Xilinx Resources 48

References 48
Revision History 49
Please Read: Important Legal Notices 49

Introduction

The Xilinx® LogiCORE™ IP CIC Compiler core provides the ability to design and implement AXI4-Stream-compliant cascaded integrator-comb (CIC) filters.

Features

- AXI4-Stream-compliant interfaces
- Decimation or interpolation
- Fixed or programmable rate change from 4 to 8192
- Three to six CIC stages
- One or two differential delays
- Support of signed, twos complement input data from 2 bits to 24 bits
- Full or limited precision output data
- Single or multichannel support for up to 16 channels
- Hardware folding for small footprint implementations
- Optional mapping to DSP48E1 Slices
- Synchronous clear input
- Clock enable input
- Use with Xilinx Vivado® IP Catalog and Xilinx System Generator for DSP

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families UltraScale™ Architecture Zynq®-7000 All Programmable 7 Series
Supported User Interfaces	AXI4-Stream
Resources	Performance and Resource Utilization web page.
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	VHDL
Constraints File	Not Applicable
Simulation Model	Encrypted VHDL
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado Design Suite System Generator for DSP
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the IP catalog feature for this core.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Cascaded Integrator-Comb (CIC) filters, also known as Hogenauer filters, are multirate filters often used for implementing large sample rate changes in digital systems. They are typically employed in applications that have a large excess sample rate. That is, the system sample rate is much larger than the bandwidth occupied by the processed signal as in digital down converters (DDCs) and digital up converters (DUCs). Implementations of CIC filters have structures that use only adders, subtracters, and delay elements. These structures make CIC filters appealing for their hardware-efficient implementations of multirate filtering.

Licensing and Ordering Information

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

For full details about performance and resource utilization, visit Performance and Resource Utilization.

Port Descriptions

Figure 2-1 and Table 2-1 illustrate and define the schematic symbol signal names.

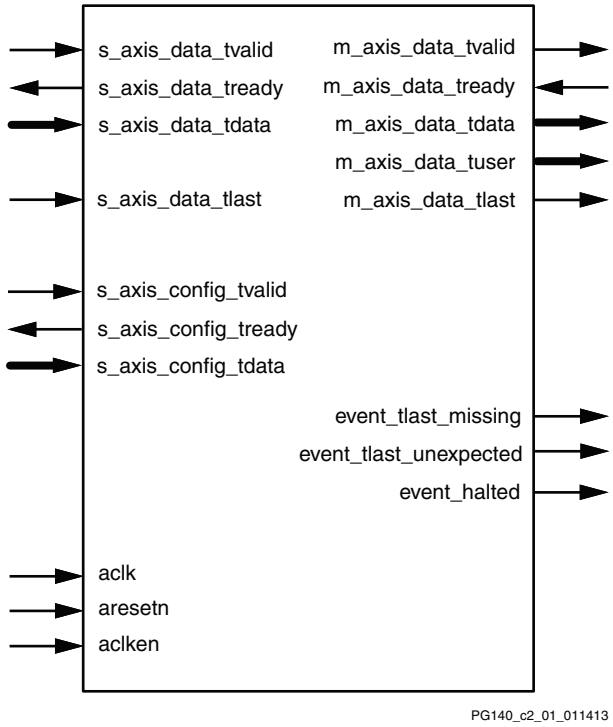

Figure 2-1: **Core Schematic Symbol**

Table 2-1: **Core Signal Pinout**

Name	Direction	Description
aclk	Input	Rising edge clock
aclken	Input	Active-High clock enable (optional).

Table 2-1: **Core Signal Pinout *(Cont'd)***

Name	Direction	Description
aresetn	Input	Active-Low synchronous clear (optional, always take priority over aclken). A minimum aresetn active pulse of two cycles is required.
s_axis_config_tvalid	Input	TVALID for the Configuration Channel. Asserted by the external master to signal that it is able to provide data.
s_axis_config_tready	Output	TREADY for the Configuration Channel. Asserted by the CIC Compiler to signal that it is ready to accept configuration data.
s_axis_config_tdata	Input	TDATA for the Configuration Channel. Carries the configuration information: RATE.
s_axis_data_tvalid	Input	TVALID for the Data Input Channel. Used by the external master to signal that it is able to provide data.
s_axis_data_tready	Output	TREADY for the Data Input Channel. Used by the CIC Compiler to signal that it is ready to accept data.
s_axis_data_tdata	Input	TDATA for the Data Input Channel. Carries the unprocessed sample data.
s_axis_data_tlast	Input	TLAST for the Data Input Channel.Only present in multichannel mode. Asserted by the external master on the sample corresponding to the last channel. This is not used by the CIC Compiler except to generate the events event_tlast_unexpected and event_tlast_missing events
m_axis_data_tvalid	Output	TVALID for the Data Output Channel. Asserted by the CIC Compiler to signal that it is able to provide sample data.
m_axis_data_tready	Input	TREADY for the Data Output Channel. Asserted by the external slave to signal that it is ready to accept data. Only present when the XCO parameter HAS_DOUT_TREADY is TRUE.
m_axis_data_tdata	Output	TDATA for the Data Output Channel.Carries the processed sample data
m_axis_data_tuser	Output	TUSER for the Data Output Channel.Only present in multichannel mode. Carries additional per-sample information: CHAN_OUT CHAN_SYNC
m_axis_data_tlast	Output	TLAST for the Data Output Channel. Only present in multichannel mode. Asserted by the CIC Compiler on the sample corresponding to the last channel.
event_tlast_missing	Output	Asserted when s_axis_data_tlast is not asserted on the data sample corresponding to the last channel. Only present in multichannel mode.
event_tlast_unexpected	Output	Asserted when s_axis_data_tlast is asserted on a data sample that does not correspond to the last channel. Only present in multichannel mode.
event_halted	Output	Asserted when the CIC Compiler tries to write data to the Data Output Channel, and it is unable to do so. Only present when the XCO HAS_DOUT_TREADY is TRUE.

Event Signals

The CIC Compiler core provides some real-time non-AXI signals to report information about the core status. These event signals are updated on a clock-cycle-by-clock-cycle basis, and are intended for use by reactive components such as interrupt controllers. These signals are not optionally configurable from the Vivado Integrated Design Environment (IDE), but are removed by synthesis tools if left unconnected.

event_tlast_missing

This event signal is asserted for a single clock cycle when `s_axis_data_tlast` is Low on the data sample corresponding to the last channel. This is intended to show a mismatch between the CIC Compiler and the upstream data source with regard to the channel synchronisation. The event pin is only available when the core is configured to have multiple channels.

event_tlast_unexpected

This event signal is asserted for a single clock cycle when the CIC Compiler sees `s_axis_data_tlast` High on any incoming data sample that does not correspond to the last channel.

This signal is intended to show a mismatch between the CIC Compiler and the upstream data source with regard to channel synchronisation.

If there are multiple unexpected highs on `s_axis_data_tlast`, then this signal is asserted for each of them. The event pin is only available when the core is configured to have multiple channels.

event_halted

This event is asserted on every cycle where the CIC Compiler needs to write data to the Data Output Channel but cannot because the buffers in the channel are full. When this occurs, the CIC Compiler core is halted and all activity stops until space is available in the channel buffers.

The event pin is only available when HAS_DOUT_TREADY is TRUE.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

This description of the CIC decimator and interpolator is based closely on that provided in *An Economical Class of Digital Filters for Decimation and Interpolation*, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-29, No. 2, April 1981 [Ref 1]. The general concept of a CIC filter is the low-pass response that results from filtering an input signal with a cascade of *N* unit-amplitude, rectangular windows of length *R*M*. The system response of such filter is:

$$H(z) = [\sum_{k=0}^{R*M-1} z^{-k}]^N$$

or

$$H(z) = \frac{(1 - z^{-R*M})^N}{(1 - z^{-1})^N} \qquad \text{Equation 3-1}$$

where

N is the number of CIC stages

R is the rate change (decimation or interpolation)

M is the differential delay in the comb section stages of the filter

The implementation of this filter response with a clever combination of comb filter sections, integrator sections, and up-sampling (for interpolation) and down-sampling (for decimation) gives rise to the hardware-efficient implementation of CIC filters.

Frequency Response Characteristics

The frequency response of a CIC filter is obtained by evaluating Equation 3-1 at:

$$z = e^{2j\pi f} \qquad \textit{Equation 3-2}$$

Where *f* is the discrete-time frequency, normalized to the *higher* frequency in a rate changing filter — input sampling frequency in a CIC decimation, or output sampling frequency in a CIC interpolator. Evaluating Equation 3-1 in the z-plane at the sample points defined by Equation 3-2 gives a magnitude frequency response as shown in Equation 3-3.

$$\left|H(f)\right| = \left[\frac{\sin(\pi RMf)}{\sin(\pi f)}\right]^N \qquad \textit{Equation 3-3}$$

This magnitude response is low-pass. In the design process of a CIC filter implementation, the parameters *R*, *M*, and *N* are selected to provide adequate passband characteristics over the frequency range from zero to a predetermined cutoff frequency *fc*. This passband frequency range is typically the bandwidth of interest occupied by the signal undergoing processing by the CIC filter. Figure 1 shows the frequency response of a 3-stage (*N* = 3) CIC filter with unity differential delay (*M* = 1) and a sample rate change *R* = 7.

According to Equation 3-3 and as seen in Figure 3-1, there are nulls in the magnitude response (transfer function zeros) at integer multiples of *f* =1/(RM). Thus, the differential delay parameter, *M*, can be used as a design parameter to control the placement of the nulls.

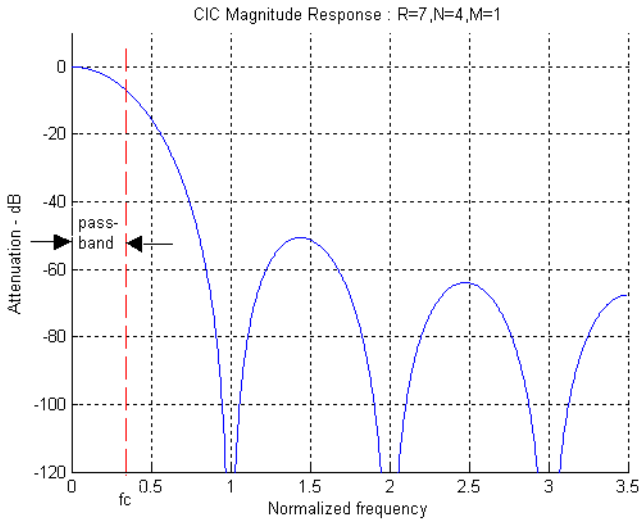

Figure 3-1: **CIC Magnitude Response**

Figure 3-2 shows the effect of the differential delay *M* on the magnitude response of a filter with three stages (*N* = 3) and a sample rate change *R* = 7. Besides the effect on the

placement of the response nulls, increasing *M* also increases the amount of attenuation in side lobes of the magnitude response.

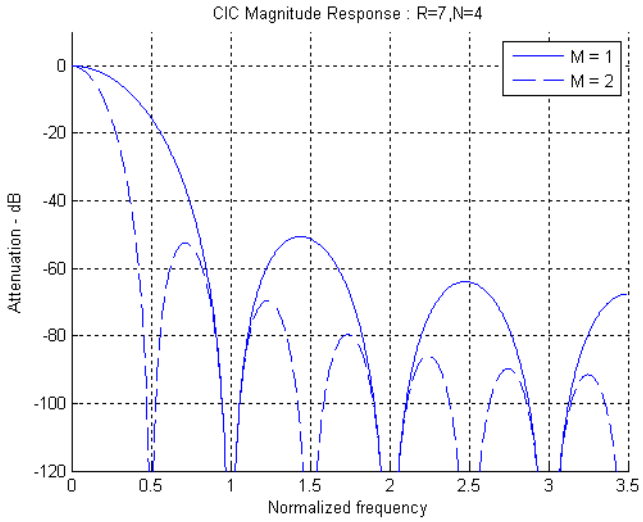

Figure 3-2: **CIC Magnitude Response – Effect of Differential Delay M**

The rate change parameter *R* can also be used to control the frequency response of the CIC filter. The effect of *R* on the magnitude response can be seen in Figure 3-3. In essence, increasing the rate change increases the length of the cascaded unit-amplitude, rectangular window of length *R*M*. This results in an increase in attenuation and decrease of the width of the response side lobes.

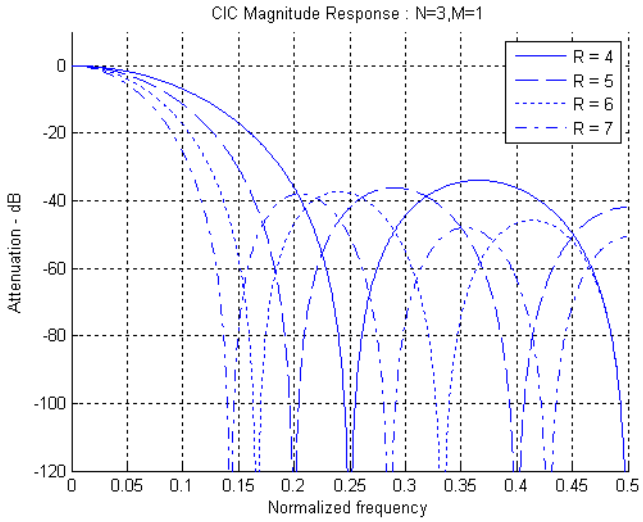

Figure 3-3: **CIC Magnitude Response – Effect of Rate Change R**

The number of stages parameters, *N*, can also be used to affect the CIC filter magnitude response. This effect can be understood from the fundamental concept of a cascade of *N*

filtering stages, each with an impulse response of a unit-amplitude, rectangular window. The larger the number of cascaded stages, the more attenuated the magnitude response side lobes become. This can be seen in Figure 3-4.

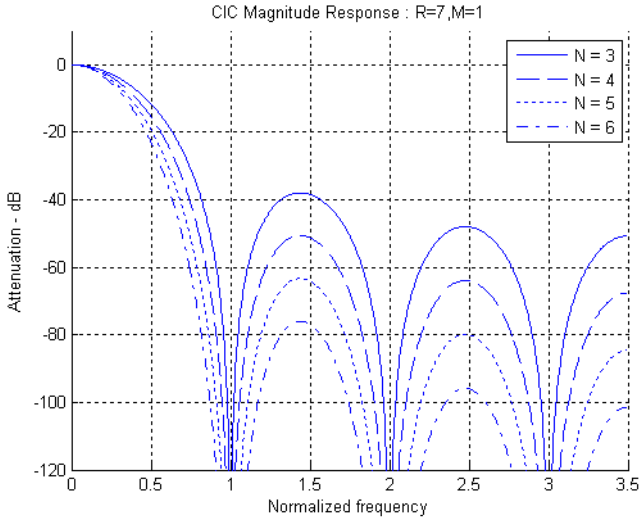

Figure 3-4: **CIC Magnitude Response – Effect of Number of Stages N**

Increasing *N* has the effect of increasing the order of the zeros in the frequency response. This, in turn, increases the attenuation at frequencies in the locality of the zero. This effect is clearly illustrated in Figure 3-4 where there is increasing attenuation of the filter side lobes as *N* is increased.

As the order of the zeros increase, the passband droop also increases, thus narrowing the filter bandwidth. The increased droop might not be acceptable in some applications. The droop is frequently corrected using an additional (non-CIC-based) stage of filtering after the CIC decimator. In the case of a CIC interpolator, the signal can be pre-compensated to account for the impact in the passband as the signal is up-sampled by the CIC interpolator.

A compensation filter (not part of the CIC Compiler) can be used to flatten the passband frequency response. For a CIC decimator, the compensation filter operates at the decimated sample rate. The compensation filter provides $(x/ \sin (x))^N$ shaping. An example of a third order $(N = 3)$ $R = 64$ compensated CIC system is shown in Figure 3-5. The plot shows the uncompensated CIC frequency response, the compensation filter frequency response, and the compensated CIC. In this case, because the number of CIC stages is three, the compensation filter has a cubic response of the form $(x/\sin (x))^3$.

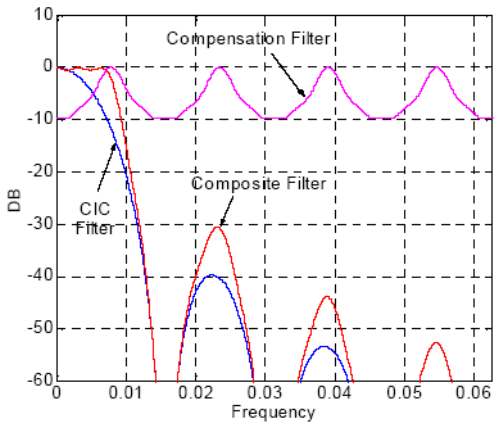

Figure 3-5: **CIC Droop Compensation**

The compensation filter coefficients employed were [–1, 4, 16, 32, –64, 136, –352, 1312, –352, 136, –64, 32, –16, 4, –1]. Figure 3-6 provides an exploded view of the compensated filter passband.

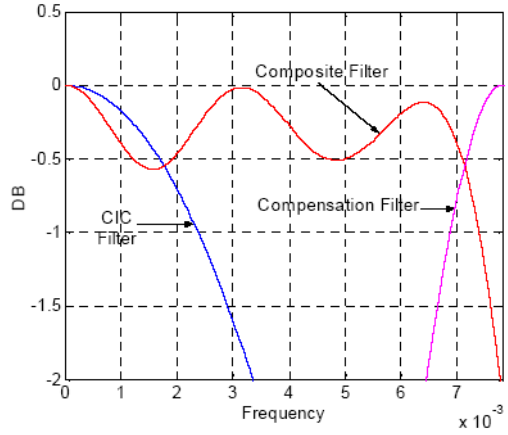

Figure 3-6: **CIC Droop Compensation – Exploded View**

CIC Decimator

When the output of the filter given by Equation 3-1 is decimated (down-sampled) by a factor *R*, the response of the filter referenced to the lower, down-sampled output rate is expressed in Equation 3-4 as:

$$H(z) = \frac{(1 - z^{-M})^N}{(1 - z^{-1})^N} \qquad \textit{Equation 3-4}$$

This response can be viewed as a cascade of *N* integrators and *N* comb filters.

$$H(z) = \frac{1}{(1-z^{-1})^N} * (1-z^{-M})^N \qquad \text{Equation 3-5}$$

A block diagram of a realization of this response can be seen in Figure 3-7. There are two sections to the CIC decimator filter: an integrator section with *N* integrator stages that processes input data samples at a sampling rate *fs*, and a comb section that operates at the lower sampling rate *fs / R*. This comb section consists of *N* comb stages with a differential delay of *M* samples per stage. The down sampling operation decimates the output of the integrator section by passing only every *Rth* sample to the comb section of the filter.

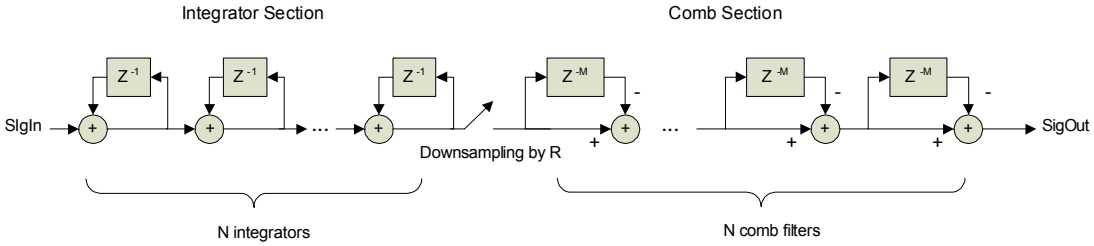

Figure 3-7: **CIC Decimation Filter**

$$\frac{k}{RM} \pm f_c, k = 1,2,... \left\lfloor \frac{R}{2} \right\rfloor \qquad \text{Equation 3-6}$$

Referring back to Figure 3-1, when the CIC filter is employed as a decimator, the frequency bands in the interval alias back into the filter passband. Care must be taken to ensure that the integrated side lobe levels do not impact the intended application. Figure 3-8 shows an example of a CIC decimator response prior to down-sampling to help illustrate the effect of aliasing. In Figure 3-8, the ideal response of a decimator with sampling rate change of *R* = 8, number of stages *N* = 3, and differential delay *M* = 1 is shown. The spectrum of the decimator input is also shown containing energy in the intended passband (low frequencies up to a cutoff frequency fc = 1/32 cycles/sample) and in the stopband (around 1/4 cycles/sample). The output of the decimator (without down-sampling) is shown to demonstrate the attenuation produced by this CIC filter. The dashed vertical lines in Figure 3-8 indicate the frequency ranges that alias to the passband when down-sampling. In this figure, the frequency axis is normalized to the (higher) sampling frequency prior to down-sampling.

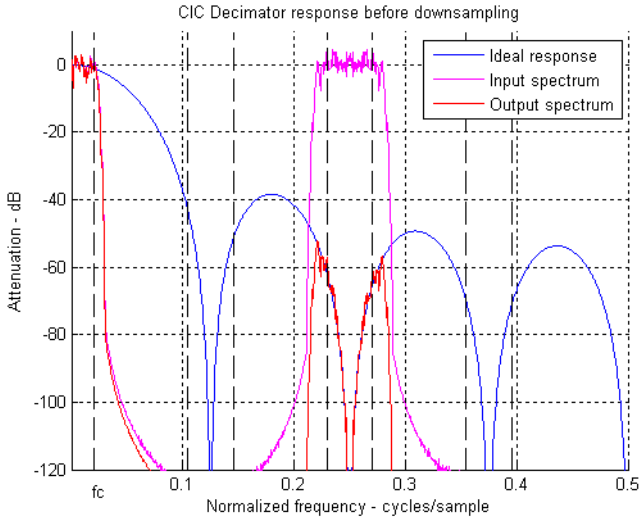

Figure 3-8: **CIC Decimator Response before Down-sampling**

Figure 3-9 shows the output spectrum of the CIC decimator example. In Figure 3-9, the frequency axis is normalized relative to the lower sampling rate obtained after down-sampling. Because of this re-normalization of frequencies, the plots in Figure 3-9 can be conceptualized as a zoomed view of the frequency range from 0 to 1/(2*R) = 1/16 cycles/sample of Figure 3-8.

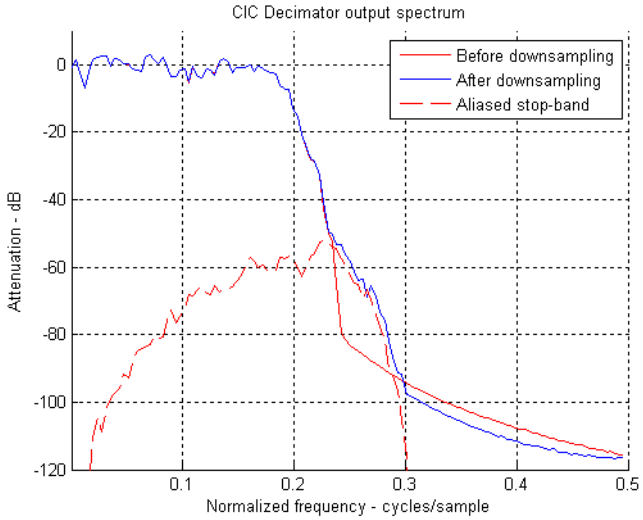

Figure 3-9: **CIC Decimator Output Spectrum**

The important points to note from Figure 3-9 are:

- The solid red plot shows the CIC output spectrum if no aliasing occurred.

- The dashed red plot shows the stopband output spectrum when aliased due to down-sampling. This aliased spectrum affects the final output of the CIC decimator by contributing additively to the output spectrum.
- The solid blue plot is the actual output of the CIC decimator which clearly shows the contribution of the aliased spectrum from down-sampling.

Again, care must be taken to ensure that the CIC decimator parameters are properly chosen to avoid detrimental effects from aliasing.

Pipelined CIC Decimator

To support high system clock frequencies, the CIC decimator is implemented using the pipelined architecture shown in Figure 3-10.

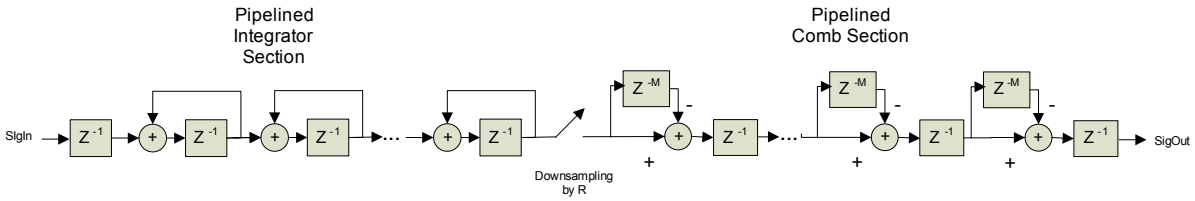

Figure 3-10: **Pipelined CIC Decimator**

The CIC datapath undergoes internal register growth that is a function of all the design parameters: *N*, *M*, *R* in addition to the input sample precision *B*. As shown in *An Economical Class of Digital Filters for Decimation and Interpolation* [Ref 1], the output bit width of a CIC decimator with full precision is given by:

$$B_{\max} = \lceil N \log_2 RM + B \rceil \qquad \text{Equation 3-7}$$

where $\lceil \ \rceil$ denotes the ceiling operator. The CIC Compiler supports both full and limited precision output. For full precision, the CIC decimator implementation uses B_{max} bits internally for each of the integrator and differentiator stages. This introduces no quantization error at the output. For limited precision (that is, output bit width less than B_{max}), the registers in the integrator and comb stages are sized to limit the quantization noise variance at the output as described in *An Economical Class of Digital Filters for Decimation and Interpolation* [Ref 1]. Consequently, the hardware resources in a CIC decimator implementation can be reduced when using limited precision output at the cost of quantization noise. This ability to trade off resources and quantization noise is important to achieve an optimum implementation.

CIC Interpolator

The structure for a CIC interpolator filter is shown in Figure 3-11. This structure is similar to that of a CIC decimator with the integrator and comb sections in reverse order. In this case, there is an up-sampling of data by a factor, *R*, between the comb and integrator sections. This rate increase is done by inserting *R-1* zero-valued samples between consecutive

samples of the comb section output. The up-sampled and filtered data stream is output at the sample rate *fs*.

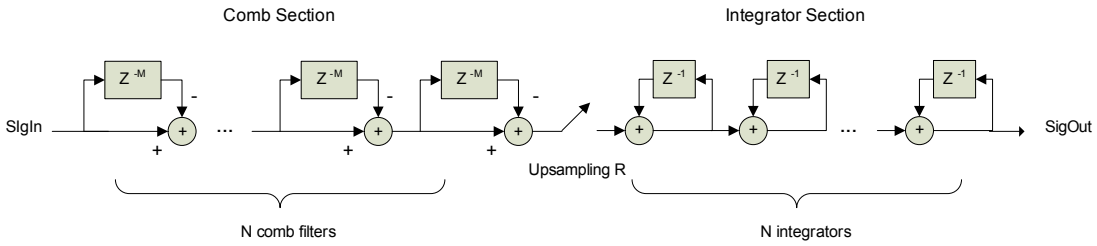

Figure 3-11: **CIC Interpolator**

For interpolation, the response of the CIC filter is applied to the up-sampled (zero-valued samples inserted) input signal. The effect of this processing is shown in Figure 3-12 in a filter with rate change *R* = 7, number of stages *N* = 4, and differential delay *M* = 1. The peaks in the output interpolated signal show the effect of the magnitude response of the CIC filter applied to the spectrum images of the up-sampled input signal.

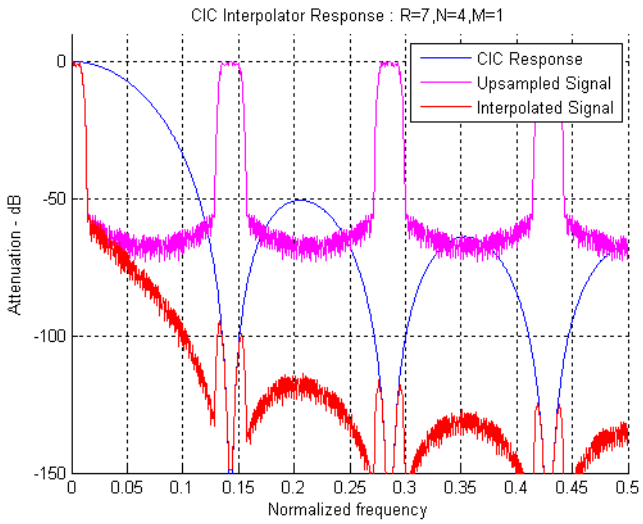

Figure 3-12: **CIC Interpolator Response**

Pipelined CIC Interpolator

Similarly to the CIC decimator, the CIC interpolator core implementation uses a pipelined structure to support high system clock frequencies. This pipelined structure is shown in Figure 3-13.

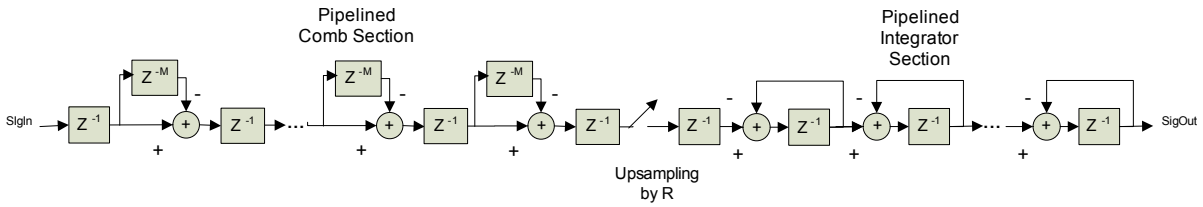

Figure 3-13: **Pipelined CIC Interpolator**

Register Growth in CIC Interpolator

The datapath in a CIC interpolator also undergoes internal register growth that is a function of all the design parameters: *N*, *M*, *R*, in addition to the input sample precision *B*. As shown in *An Economical Class of Digital Filters for Decimation and Interpolation* [Ref 1], the registers in the comb and integrator sections grow monotonically with the maximum register size occurring at the output of the last stage (output of the CIC filter). The maximum register width is given by Equation 3-8:

$$B_{\max} = \left\lceil \log_2 \frac{(RM)^N}{R} + B \right\rceil \qquad \text{Equation 3-8}$$

where $\lceil\ \rceil$ denotes the ceiling operator. The CIC Compiler always sizes the internal stage registers according to the register growth as described in *An Economical Class of Digital Filters for Decimation and Interpolation* [Ref 1]. The output of the filter can be selected to be full or limited precision (with truncation or rounding) to accommodate an output width specific to an application. Using limited precision does not affect the internal register sizes and only the final stage output is scaled, and rounded if desired, to provide the selected output width.

Output Width and Gain

As illustrated by Equation 3-7 and Equation 3-8, the gain of a CIC filter is a function of all the key design parameters. When the output width is equal to the maximum register width, the core outputs the full precision result and the magnitude of the core output reflects the filter gain. When the output width is set to less than the maximum register width, the output is truncated with a corresponding reduction in gain.

When the core is configured to have a programmable rate change, there is a corresponding change in gain as the filter rate is changed. When the output is specified to full precision, the change in gain is apparent in the core output magnitude as the rate is changed. When the output is truncated, the core shifts the internal result, given the B_{max} for the current rate change, to fully occupy the output bits.

Clocking

The core uses a single clock for all functions and interfaces.

Resets

The core uses a single active-Low reset signal for all functions and interfaces.

Protocol Description

Control Signals and Timing

Symbol data to be processed is loaded into the CIC Compiler core using the Data Input Channel. Processed symbol data is unloaded using the Data Output Channel. Both of these use the AXI4-Stream protocol. Figure 3-14 shows the basics of this protocol.

TVALID is driven by the channel master to show that it has data to transfer, and TREADY is driven by the channel slave to show that it is ready to accept data. When both TVALID and TREADY are High, a transfer takes place. Points A in the diagram show clock cycles where no data is transferred because neither the master or the slave is ready. Point B shows two clock cycles where data is not transferred because the Master does not have any data to transfer. This is known as a master waitstate. Point C shows a clock cycle where no data is transferred because the slave is not ready to accept data. This is known as a slave waitstate. Master and slave waitstates can extend for any number of clock cycles.

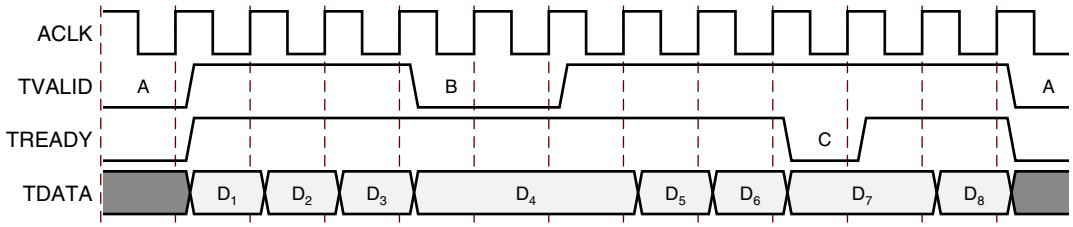

Figure 3-14: **AXI Transfers and Terminology**

When the master asserts TVALID High, it must remain asserted (and the associated data remain stable) until the slave asserts TREADY High.

Figure 3-14 shows the loading of 8 samples. The upstream master drives TVALID and the CIC Compiler drives TREADY. In this case, both the master and the CIC Compiler insert waitstates.

Figure 3-14 also shows the unloading of 8 samples. The CIC Compiler drives TVALID and the downstream slave drives TREADY. In this case, both the CIC Compiler and the slave insert waitstates. This only applies when the core is configured to have a TREADY port on the Data Output Channel (XCO HAS_DOUT_TREADY = TRUE). When this is false, there is no TREADY signal on the Data Output Channel and the downstream slave cannot insert waitstates. The slave must be able to respond immediately on every clock cycle where the CIC Compiler produces data (`m_axis_data_tvalid` asserted High). If the slave cannot respond immediately, then data is lost.

For multiple-channel implementations, the CIC Compiler core supports time-multiplexed input and output. The filter input data in the DATA field of the Data Input Channel TDATA vector (`s_axis_data_tdata`) is expected to have an ordered, time-multiplexed format. The core produces time-multiplexed output data on the DATA field of the Data Output Channel TDATA vector (`m_axis_data_tdata`). Two additional fields are included in multichannel implementation. The CHAN_SYNC field in the Data Output Channel TUSER vector (`m_axis_data_tuser`) indicates the output corresponding to the first channel in the time-multiplexed stream. The CHAN_OUT field in the Data Output Channel TUSER vector (`m_axis_data_tuser`) contains the channel number for each output in the time-multiplexed steam.

For programmable rate implementations, the RATE field in the Configuration Channel TDATA vector (`s_axis_config_tdata`) controls the rate change in the CIC Compiler filter core. The RATE field is sampled when `s_axis_config_tvalid` and `s_axis_config_tready` are both asserted High. The core uses the new RATE value on the next input sample, for a single channel implementation, or the next input to the first channel, for multiple channel implementations.

All of the waveforms (Figure 3-15 to Figure 3-24) are shown with HAS_DOUT_TREADY = FALSE. Setting this to TRUE allows the downstream data slave to delay the data output of the CIC Compiler. It also allows the Data Input Channel to buffer samples so that they can be supplied at a faster rate than the core can process them.

To simplify the waveforms, the following field aliases are used:

- DIN is used to represent the DATA field in the Data Input Channel TDATA vector (`s_axis_data_tdata`)
- DOUT is used to represent the DATA field in the Data Output Channel TDATA vector (`m_axis_data_tdata`)
- CHAN_SYNC is used to represent the CHAN_SYNC field in the Data Output Channel TUSER vector (`m_axis_data_tuser`)
- CHAN_OUT is used to represent the CHAN_OUT field in the Data Output Channel TUSER vector (`m_axis_data_tuser`)
- RATE is used to represent the RATE field in the Configuration Channel TDATA vector (`s_axis_config_tdata`)

Decimator

The timing for a CIC decimator with a down-sampling factor *R* = 4 is shown in Figure 3-15. In this example, the core is not oversampled and can accept a new input sample on every clock edge. Some number of clock cycles after the first input sample has been written to the filter, `m_axis_data_tvalid` is asserted by the filter to indicate that the first output sample is available. This time interval is a function of the down-sampling factor *R* and a fixed latency that is related to internal pipeline registers in the core. The number of pipeline stages depends on the core parameters. After the first output sample has been produced, subsequent outputs are available every *R* clock cycles.

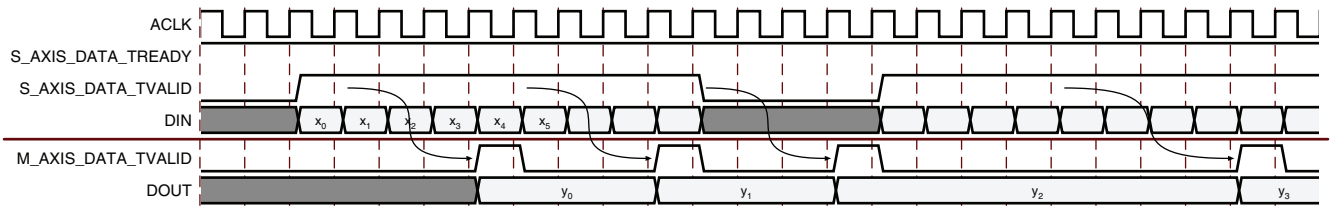

Figure 3-15: **CIC Decimator – Fixed Rate, Single Channel**

Figure 3-16 shows the timing for the same filter configuration with an input sample period of 3. At point A in the waveform, the CIC Compiler is ready to accept data but the master does not provide it. The CIC Compiler continues to ask until it is provided (point B). At point C in the waveform, the master provides data before the CIC Compiler requests it. The master has to continue supplying this data until the CIC Compiler accepts it (at point D).

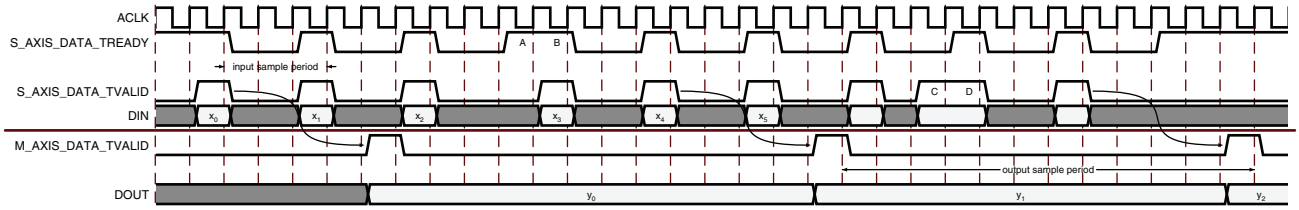

Figure 3-16: **CIC Decimator – Fixed Rate, Single Channel, Oversampled**

Multichannel Decimators can be configured to produce data in two timing modes, Block and Streaming:

- Block mode: samples for the channels are produced back-to-back. That is, the data for channel N+1 is produced immediately after the data for channel N.
- Streaming mode: samples for the channels are produced evenly over the entire sample period.

See Figure 3-18 and Figure 3-19 for more information.

These modes operate independently of the AXI4 interface and they refer to the part of the core that processes the data. When HAS_DOUT_TREADY = 1 the AXI4 interface can buffer

data in the Data Output Channel which means that streaming mode can start to behave like block mode. If the downstream system does not consume data when it first becomes available, the Data Output Channel can start to fill. In this case, the Data Output Channel produces back-to-back data (using `m_axis_data_tvalid`) until the buffer in the channel is empty, even though the processing part of the core did not produce it back-to-back.

Figure 3-17 shows the timing for a multichannel CIC decimator with a rate change $R = 4$. In this example the decimator filter handles three channels of data and is configured to use the block-based interface. The input to the decimator DIN shows the time-multiplexed samples with labels to indicate the corresponding channel number. The output of the decimator DOUT shows the time-multiplexed data.

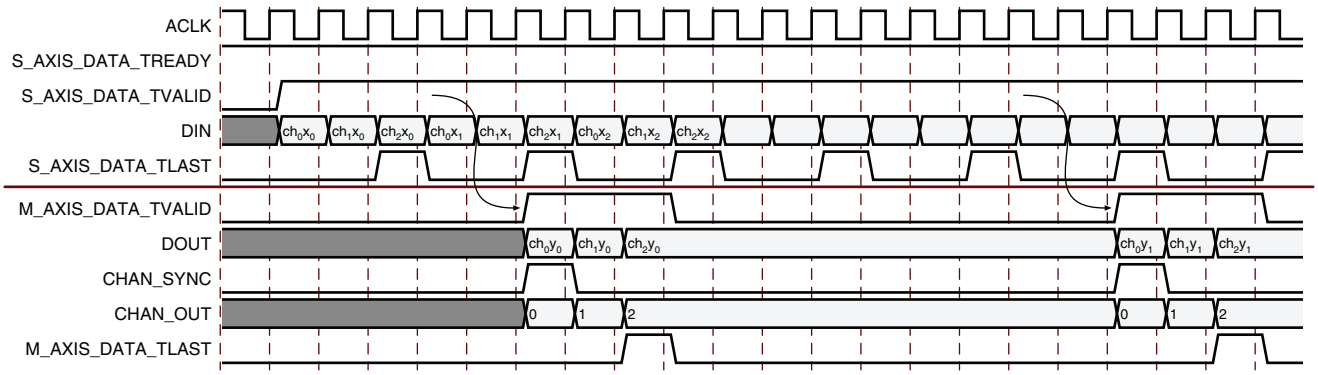

Figure 3-17: **CIC Decimator – Fixed Rate, Multichannel, Block interface**

Figure 3-18 shows the timing for the same filter configuration using the streaming interface.

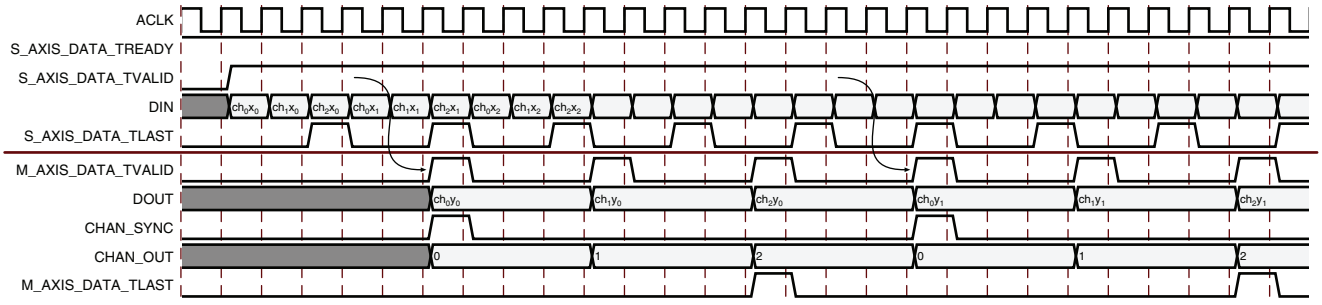

Figure 3-18: **CIC Decimator – Fixed Rate, Multichannel, Streaming interface**

Figure 3-19 shows the timing for a CIC decimator with programmable rate. In the timing diagram, the decimator is shown with an initial down-sampling rate value of 4. After some time, the down sampling rate is changed to 7 by setting the value in the RATE field to 7 and asserting `s_axis_config_tvalid` at point A in the waveform. As the rate is only applied when the next sample is accepted by the CIC Compiler, `s_axis_config_tready` deasserts for a cycle while the rate change is applied. This prevents the upstream master providing a new rate which cannot be accepted by the core.

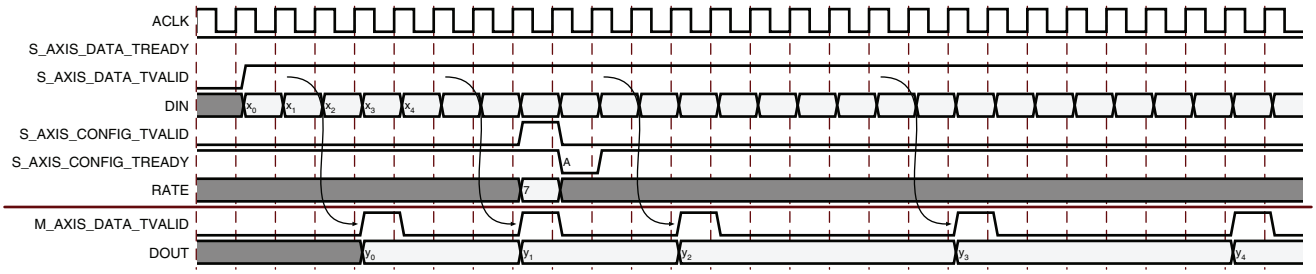

Figure 3-19: **CIC Decimator with Programmable Rate**

Interpolator

Figure 3-20 shows the timing for a CIC interpolator with an up-sampling factor *R* = 4. A new input sample can be accepted by the core every fourth cycle of the clock. After the initial start-up latency, `m_axis_data_tvalid` is asserted, and a new filter output is available on every subsequent clock edge. For every input delivered to the filter core, four output samples are generated. At point A in the waveform, the master provides data before the CIC Compiler requests it. The master has to continue supplying this data until the CIC Compiler accepts it (at point B). At point C in the waveform, the CIC Compiler is ready to accept data but the master does not provide it. The CIC Compiler continues to ask until it is provided (point D).

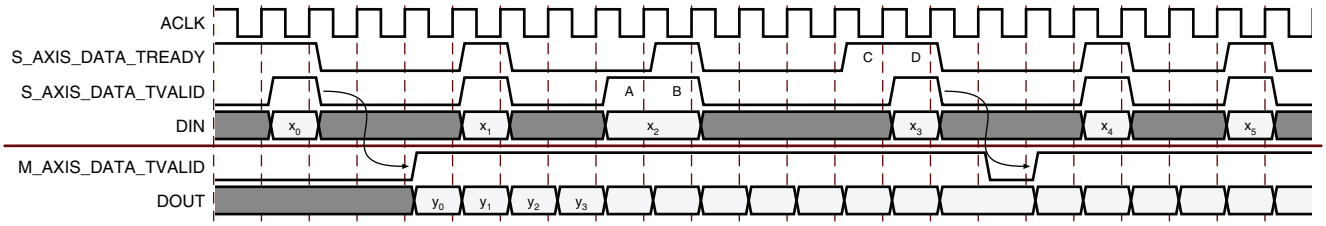

Figure 3-20: **CIC Interpolator – Fixed Rate, Single Channel**

Figure 3-21 shows the same filter configuration with an input sample period of 8.

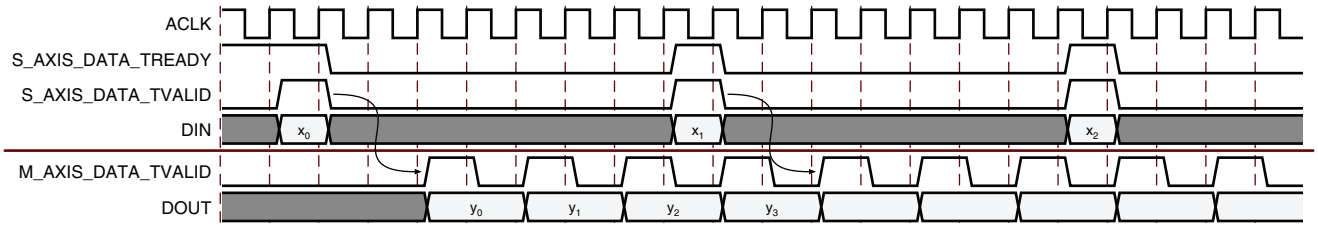

Figure 3-21: **CIC Interpolator – Fixed Rate, Single Channel, Oversampled**

Multichannel Interpolators can be configured to consume data in two timing modes, Block and Streaming:

- Block mode: samples for the channels are consumed back-to-back. That is, the data for channel N+1 is consumed immediately after the data for channel N.
- Streaming mode: samples for the channels are consumed evenly over the entire sample period

See Figure 3-22 and Figure 3-23 for more information.

These modes operate independently of the AXI4 interface and they refer to the part of the core that processes the data. When HAS_DOUT_TREADY = 1 the AXI4 interface can buffer data in the Data Input Channel which means that streaming mode might start to behave like block mode. Until its buffer is full, the Data Input Channel requests back-to-back data (using `s_axis_data_tready`) even though the processing part of the core does not consume it immediately.

Figure 3-22 shows the timing for a multichannel CIC interpolator with a rate change *R* = 4. In this example the interpolator filter handles two channels of data and uses the block-based interface. The input DIN shows the time-multiplexed samples with labels to indicate the corresponding channel number. The output DOUT shows the time-multiplexed data samples.

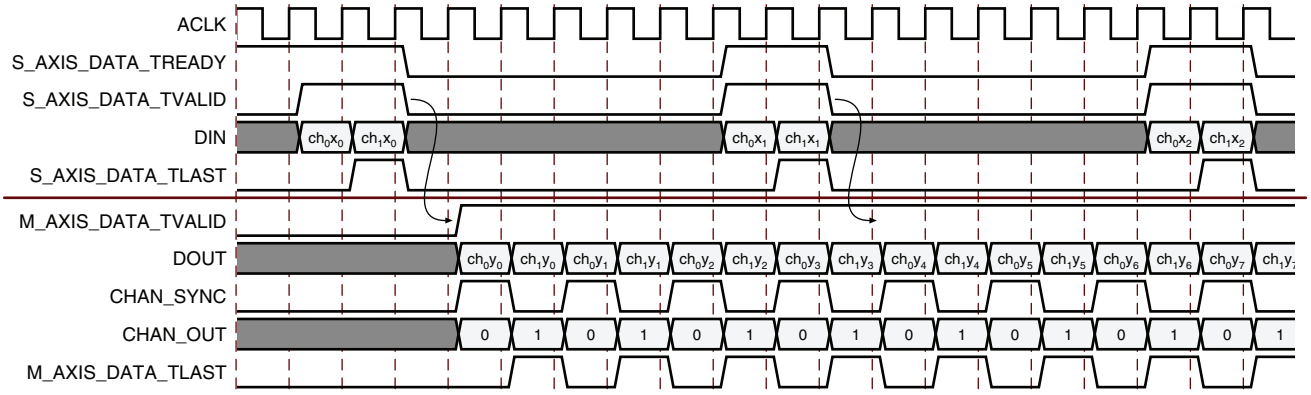

Figure 3-22: **CIC Interpolator – Fixed Rate, Multichannel, Block interface**

Figure 3-23 shows the same filter configuration using the streaming interface.

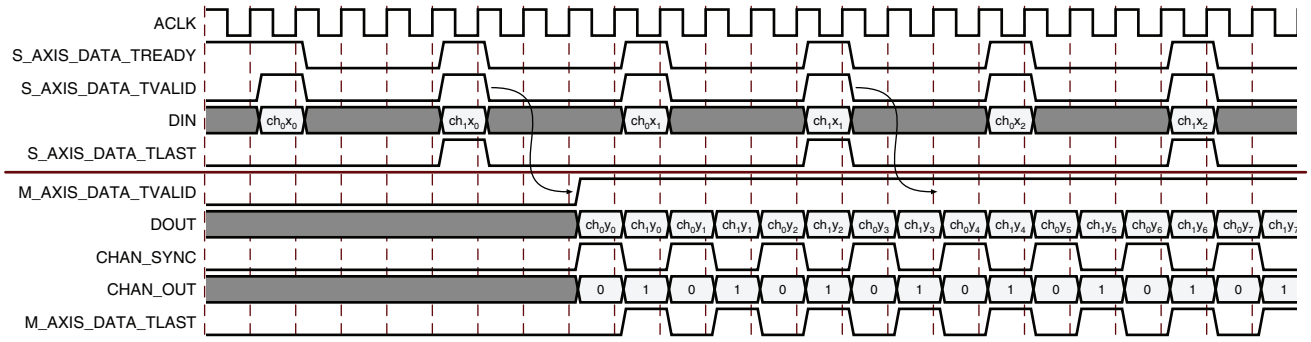

Figure 3-23: **CIC Interpolator – Fixed Rate, Multichannel, Streaming interface**

Figure 3-24 shows the timing for a CIC interpolator with programmable rate. In the timing diagram, the interpolator is shown with an initial up-sampling rate value of 4. After some time, the up-sampling rate is changed to 7 by setting the RATE field to 7 and asserting `s_axis_config_tvalid` at point A in the waveform. As the rate is only applied when the next sample for the first channel is accepted by the CIC Compiler, `s_axis_config_tready` deasserts until the rate change is applied (point B). In this example, the sample (x_1) following point A is for the second channel, so the rate is not applied here. This prevents the upstream master providing a new rate which cannot be accepted by the core.

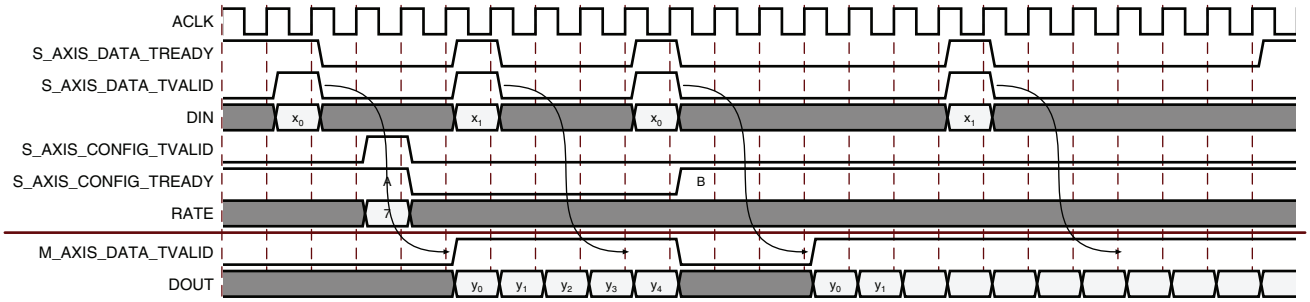

Figure 3-24: **CIC Interpolator with Programmable Rate**

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 2]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 2] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4].

Core Parameters

The CIC Compiler core customization screen in the Vivado IDE has three pages used to configure the core plus three informational/analysis tabs.

IMPORTANT: *The component name at the top of the GUI is the name of the core component to be instantiated. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and "_".*

Tab 1: IP Symbol

The IP Symbol tab illustrates the core pinout.

Tab 2: Frequency Response

The Freq. Response tab displays the filter frequency response (magnitude only).

IMPORTANT: *The frequency axis in this plot is normalized frequency.*

Although the values in the Vivado IDE plot range from 0 to 1.0, they represent the range from 0 to 1/2 the sampling frequency. It is also important to note that the normalizing sampling frequency implied in the plot depends on the type of filter. For a CIC decimator, the normalizing sampling frequency is the higher input sampling frequency. For a CIC interpolator, the normalizing frequency is the higher output sampling frequency.

- **Response Magnitude**: Specifies the magnitude scaling of the frequency response: Normalized; Full Precision (the absolute filter gain) and Output Quantization (the effective filter gain given the core output width). In previous versions of the core, the frequency response was always normalized. All plots shown in General Design Guidelines use normalized magnitude.
- **Passband Range**: Two fields are available to specify the passband range, the left-most being the minimum value and the right-most the maximum value. The values are specified in the same units as on the graph x-axis (for example, normalized to pi radians per second). For the specified range, the passband maximum, minimum and ripple values are calculated and displayed (in dB).
- **Stopband Range**: Two fields are available to specify the stopband range, the left-most being the minimum value and the right-most the maximum value. The values are specified in the same units as on the graph x-axis (for example, normalized to pi radians per second). For the specified range, the stopband maximum value is calculated and displayed (in dB).

Note: Any range can be specified for the passband or stopband, allowing closer analysis of any region of the response.

Tab 3: Implementation Details

Resource Estimates

Based on the options selected, this field displays the DSP48E1 slice count and 18K block RAM numbers. The resource numbers are an estimate; for exact resource usage, and slice/LUT-FlipFlop pair information, a MAP report should be consulted.

AXI4-Stream Port Structure

This section shows how the CIC Compiler fields are mapped to the AXI4 channels.

Filter Options

- **Filter Type**: The CIC Compiler core supports both interpolator and decimator types. When the filter type is selected as *decimator*, the input sample stream is down-sampled by the factor *R*. When an *interpolator* is selected, the input sample is up-sampled by *R*.
- **Number of Stages**: Number of integrator and comb stages. If *N* stages are specified, there are *N* integrators and *N* comb stages in the filter. The valid range for this parameter is 3 to 6.
- **Differential Delay**: Number of unit delays employed in each comb filter in the comb section of either a decimator or interpolator. The valid range of this parameter is 1 or 2.
- **Number of Channels**: Number of channels to support in implementation. Valid values for this parameter are 1 to 16, 32, 64 and 128.
- **Fixed/Programmable**: Type of rate change is fixed or programmable.
- **Fixed or Initial Rate**: Rate change factor (for fixed type) or initial rate change factor (for programmable type). For an interpolation filter, the rate change specifies the amount of up-sampling. For a decimator, it specifies the amount of down-sampling.
- **Minimum Rate**: Minimum rate change factor for programmable rate change.
- **Maximum Rate**: Maximum rate change factor for programmable rate change.
- **Hardware Oversampling Specification format**: Selects which format is used to specify the hardware oversampling rate, the number of clock cycles available to the core to process an input sample and generate an output. This value directly affects the level of parallelism in the core implementation and resources used. When *Frequency Specification* is selected, you need to specify the Input Sampling Frequency and Clock Frequency. The ratio between these values along with other core parameters determine the hardware oversampling rate. When *Sample Period* is selected, you need to specify the integer number of clock cycles between input samples.
- **Input Sample Frequency**: This field can be an integer or real value. It specifies the sample frequency for one channel. The upper limit is set based on the clock frequency and filter parameters such as interpolation rate and number of channels.

- **Clock Frequency**: This field can be an integer or real value. The limits are set based on the sample frequency, interpolation rate and number of channels.

  Note: This field influences architecture choices only; the specified clock rate might not be achievable by the final implementation.

- **Input Sample Period**: Integer number of clock cycles between input samples. When the multiple channels have been specified, this value should be the integer number of clock cycles between the time division multiplexed input sample data stream.

Implementation Options

- **Input Data Width**: Number of bits for input data. The valid range of this parameter is 2 to 32. In IP integrator, this parameter is auto-updated.
- **Quantization**: Type of quantization for limited precision output, Full Precision or Truncation. This quantization applies only to the output and is not applied in the intermediate stages of the CIC Compiler filter.
- **Output Data Width**: Number of bits for output data. The valid range of this parameter is up to 48 bits with the minimum value set to the input data width.
- **Use Xtreme DSP Slice**: Use DSP hardware primitive slices in the filter implementation.
- **Use Streaming Interface**: Specifies if a streaming interface is used for multiple channel implementations. See Decimator in Chapter 3, for further details.
- **Has DOUT TREADY**: Specifies if the Data Output Channel has a TREADY
- **ACLKEN**: Determines if the core has a clock enable input (`aclken`).
- **ARESETN**: Determines if the core has an active-Low synchronous clear input (`aresetn`).

  Note:

  a. The signal `aresetn` always takes priority over `aclken`, that is, `aresetn` takes effect regardless of the state of `aclken`.

  b. The signal `aresetn` is active-Low.

  c. The signal `aresetn` should be held active for at least 2 clock cycles. This is because, for performance, `aresetn` is internally registered before being fed to the reset port of primitives.

Summary

In addition to all the parameterization values of the core, the summary page displays:

- **Bits per Stage**: The number of bits used in each of the stages of the CIC Compiler filter implementation. These numbers are computed based on the register growth analysis presented in *An Economical Class of Digital Filters for Decimation and Interpolation* [Ref 1].

- **Latency**: The input to output latency in the CIC Compiler core implementation. When HAS_DOUT_TREADY is TRUE then the actual latency might be greater than reported because throughput can be controlled by the system connected to the Data Output Channel. The value reported by the Vivado Design Suite is the minimum latency.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Field Label	User Parameter	Default Value
Filter Type	filter_type	Interpolation
Number of Stages	number_of_stages	3
Differential Delay	differential_delay	1
Number of Channels	number_of_channels	1
Sample Rate	sample_rate_changes	Fixed
Fixed Or Initial Rate	fixed_or_initial_rate	4
Minimum Rate	minimum_rate	4
Maximum Rate	maximum_rate	4
Rate Specification	ratespecification	Frequency_Specification
Input Sample Frequency	input_sample_frequency	0.001
Clock Frequency	clock_frequency	200.0
Sample Period	sampleperiod	4
Input Data Width	input_data_width	18
Quantization	quantization	Full_precision
Output Data Width	output_data_width	22
Use DSP48 Slice	use_xtreme_dsp_slice	True
ACLKEN	has_aclken	False
ARESETn	has_aresetn	False
Output Tready	has_dout_tready	False

System Generator for DSP GUI

The CIC Compiler core is available through Xilinx System Generator for DSP, a design tool that enables the use of the model-based design environment, Simulink® product for FPGA design. The CIC Compiler core is one of the DSP building blocks provided in the Xilinx blockset for Simulink. The core can be found in the Xilinx Blockset in the DSP section. The block is called CIC Compiler v4.0. See the System Generator User Manual for more information.

This section describes each tab of the System Generator for DSP GUI and details the parameters that differ from the Vivado IDE. See Core Parameters for detailed information about all other parameters.

Tab 1: Filter Specification

The Filter Specification tab is used to define the basic filter configuration as on the Filter Options, page 28, in the Vivado IDE.

- **Hardware Oversampling Specification format**: Selects which method is used to specify the hardware oversampling rate. This value directly affects the level of parallelism of the core implementation and resources used. When *Maximum Possible* is selected, the core uses the maximum oversampling given the sample period of the signal connected to the Data field of the `s_axis_data_tdata` port. When *Hardware Oversampling Rate* is selected, you can specify the oversampling rate. When *Sample Period* is selected, the core clock is connected to the system clock, and the value specified for the Sample Period parameter sets the input sample rate the core supports. The Sample Period parameter also determines the hardware oversampling rate of the core. When *Sample Period* is selected, the core is forced to use the `s_axis_data_tvalid` control port. See Decimator in Chapter 3, for more details on the core control ports.
- **Sample Period**: Specifies the input sample period supported by the core.
- **Hardware Oversampling Rate**: Specifies the hardware oversampling rate to be applied to the core.

Tab 2: Implementation Options

The Implementation tab is used to define implementation options. See Implementation Options, page 29, in the Vivado IDE for details of all core parameters on this tab.

- **Has ARESETN**: Specifies if the core has a reset pin (the equivalent of selecting the Has ARESETN option in the Vivado IDE).
- **Has ACLKEN**: Specifies if the core has a clock enable pin (the equivalent of selecting the Has ACLKEN option in the Vivado IDE).
- **Has DOUT TREADY**: Specifies if the core has a TREADY pin for the Data Output Channel (the equivalent of selecting the HAS_DOUT_TREADY option in the Vivado IDE)
- **FPGA Area Estimation**: See the System Generator documentation for detailed information about this option.

Using the CIC Compiler IP Core

The Vivado IDE performs error-checking on all input parameters. Resource estimation and optimum latency information are also available.

Several files are produced when a core is generated, and customized instantiation templates for Verilog and VHDL design flows are provided in the `.veo` and `.vho` files, respectively. For detailed instructions, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

XCI Parameters

Table 4-2 defines valid entries for the XCI parameters. Parameters are not case sensitive. Default values are displayed in bold. Xilinx strongly suggests that XCI parameters are not manually edited in the XCI file; instead, use the Vivado IDE to configure the core and perform range and parameter value checking. The XCI parameters are useful for defining the interface to other Xilinx tools.

Table 4-2: **XCI Parameters**

XCI Parameter	Valid Values
Component_Name	ASCII text using characters: a...z, 0...9 and '_' ; starting with a letter
Filter_Type	**Interpolation**, Decimation
Number_Of_Stages	**3**,4,5,6
Differential_Delay	**1**, 2
Number_Of_Channels	**1**–16, 32, 64 and 128
Sample_Rate_Changes	**Fixed**, Programmable
Fixed_Or_Initial_Rate	**4**–8192
Minimum_Rate	4–8191
Maximum_Rate	5–8192
RateSpecification	**Frequency_Specification**, Sample_Period
Input_Sample_Frequency	0.000001–600.0
Clock_Frequency	0.000001–600.0
SamplePeriod	1–10000000
Input_Data_Width	2–32 Default is **18**
Quantization	**Full_Precision**, Truncation
Output_Data_Width	2–104
Use_Xtreme_DSP_Slice	false, **true**
Use_Streaming_Interface	**true**, false
HAS_DOUT_TREADY	**false**, true
HAS_ACLKEN	**false**, true
HAS_ARESETN	**false**, true

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite.

Demonstration Test Bench

When the core is generated using IP catalog, a demonstration test bench is created. This is a simple VHDL test bench that exercises the core.

The demonstration test bench source code is one VHDL file: `demo_tb/tb_<component_name>.vhd` in the IP catalog output directory. The source code is comprehensively commented.

Using the Demonstration Test Bench

The demonstration test bench instantiates the generated CIC Compiler core. Compile the netlist and the demonstration test bench into the work library (see your simulator documentation for more information on how to do this). Then simulate the demonstration test bench. View the test bench signals in your simulator waveform viewer to see the operations of the test bench.

Demonstration Test Bench in Detail

The demonstration test bench performs these tasks:

- Instantiate the core
- Generate a clock signal
- Drive the core clock enable and reset input signals (if present)
- Drive the core input signals to demonstrate core features (see the following sections for details)
- Provide signals showing the separate fields of AXI4 TDATA and TUSER signals

The demonstration test bench drives the core input signals to demonstrate the features and modes of operation of the core. The test bench drives a sine wave into the CIC core. In multichannel mode, the frequency of the sine wave increases with each channel. The output

of the core shows the same sine wave (or waves in multichannel mode) but in a decimated or interpolated format. Alias signals are used to decode the AXI4 channels and allow easy viewing of the input and output data.

The operations performed by the demonstration test bench are appropriate for the configuration of the generated core, and are a subset of the following operations (in order):

1. Asserts reset at the start of the test. Only available when the core is configured with a reset (HAS_ARESETN = TRUE).
2. Sends data to CIC Compiler core with no waitstates. The upstream master supplies data to the CIC Compiler core at the input sample rate.
3. Sends data to CIC Compiler core with waitstates. The upstream master adds random waitstates to the input samples by deasserting `s_axis_data_tvalid`.
4. Asserts and deasserts clock enable (`aclken`) on alternating clock cycles. This has the effect of halving the input sample rate. Only available when the core is configured with a clock enable (HAS_ACLKEN = TRUE)
5. Changes the rate of the core. Only available in programmable rate cores.
6. Injects waitstates on Data Output Channel. The downstream slave keeps `m_axis_data_tready` deasserted for random periods of time. Only available when the core is configured with a TREADY on the Data Output Channel (HAS_DOUT_TREADY = TRUE).

Customizing the Demonstration Test Bench

It is possible to modify the demonstration test bench to drive the core inputs with different data or to perform different operations. Input data generated on the fly using a function called `calculate_next_input_sample()`. By default it generates a sine wave, but could be modified to generate an impulse (for example).

The clock frequency of the core can be modified by changing the CLOCK_PERIOD constant.

Appendix A

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information on migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes in the XCO/XCI File

The IP catalog core update feature can be used to update an existing CIC Compiler XCO file to version v4.0 of the CIC Compiler core. The core can then be regenerated to create a new netlist. See the Vivado IP catalog documentation for more information on this feature. It should be noted that the update mechanism alone does not create a core compatible with v2.0. See Instructions for Minimum Change Migration. CIC Compiler v4.0 has additional parameters for AXI4-Stream support. Table A-1 shows the changes to XCO parameters from v3.0 and v4.0 have identical XCI parameters, ports, latency and behavior.

Table A-1: **XCO/XCI Parameter Changes from v2.0 to v4.0**

Version 2.0	Version 4.0	Notes
GUI_Behaviour	GUI_Behaviour	Unchanged
Filter_Type	Filter_Type	Unchanged
Number_of_Stages	Number_of_Stages	Unchanged
Differential_Delay	Differential_Delay	Unchanged

Table A-1: **XCO/XCI Parameter Changes from v2.0 to v4.0 *(Cont'd)***

Version 2.0	Version 4.0	Notes
Number_Of_Channels	Number_Of_Channels	Unchanged
Sample_Rate_Changes	Sample_Rate_Changes	Unchanged
Fixed_Or_Initial_Rate	Fixed_Or_Initial_Rate	Unchanged
Minimum_Rate	Minimum_Rate	Unchanged
Maximum_Rate	Maximum_Rate	Unchanged
RateSpecification	RateSpecification	Unchanged
Input_Sample_Frequency	Input_Sample_Frequency	Unchanged
Clock_Frequency	Clock_Frequency	Unchanged
HardwareOversamplingRate	HardwareOversamplingRate	Unchanged
SamplePeriod	SamplePeriod	Unchanged
Passband_Min	Passband_Min	Unchanged
Stopband_Min	Stopband_Min	Unchanged
Passband_Max	Passband_Max	Unchanged
Stopband_Max	Stopband_Max	Unchanged
Input_Data_Width	Input_Data_Width	Unchanged
Output_Data_Width	Output_Data_Width	Unchanged
Quantization	Quantization	Unchanged
CE	HAS_ACLKEN	Renamed
SCLR	HAS_ARESETN	Renamed
ND	ND	Unchanged
Use_Streaming_Interface	Use_Streaming_Interface	Unchanged
Use_Xtreme_DSP_Slice	Use_Xtreme_DSP_Slice	Unchanged
	HAS_DOUT_TREADY	New

Port Changes

Table A-2 details the changes to port naming, additional or deprecated ports and polarity changes from v2.0 to v4.0

Table A-2: **Port Changes from Version 2.0 to Version 4.0**

Version 2.0	Version 4.0	Notes
CLK	ACLK	Renamed
CE	ALCKEN	Renamed
SCLR	ARESETN	Renamed Polarity change (now active-Low) Minimum length now two clock cycles
DIN	S_AXIS_DATA_TDATA	Renamed

Table A-2: **Port Changes from Version 2.0 to Version 4.0 *(Cont'd)***

Version 2.0	Version 4.0	Notes
ND	S_AXIS_DATA_TVALID	Renamed
RFD	S_AXIS_DATA_TREADY	Renamed
	S_AXIS_DATA_TLAST	New signal
RATE	S_AXIS_CONFIG_TDATA	Renamed
RATE_WE	S_AXIS_CONFIG_TVALID	Renamed
	S_AXIS_CONFIG_TREADY	New signal
DOUT	M_AXIS_DATA_TDATA	Renamed
RDY	M_AXIS_DATA_TVALID	Renamed
	M_AXIS_DATA_TREADY	New signal Optional
	M_AXIS_DATA_TLAST	New signal
CHAN_SYNC	M_AXIS_DATA_TUSER	Renamed
CHAN_OUT	M_AXIS_DATA_TUSER	Renamed
	event_tlast_missing	New signal
	event_tlast_unexpected	New signal
	event_halted	New signal

Functionality Changes

Latency Changes

HAS_DOUT_TREADY = 0

The latency of the core is identical to that of the equivalent configuration of v2.0.

HAS_DOUT_TREADY = 1

The latency of the core is variable, so that only the minimum possible latency can be determined. The latency is a minimum of three cycles longer than for the equivalent configuration of v2.0. The update process cannot account for this and guarantee equivalent performance.

Simulation

Starting with CIC Compiler v4.0 (2013.3 version), behavioral simulation models have been replaced with IEEE P1735 Encrypted VHDL. The resulting model is bit and cycle accurate with the final netlist. For more information on simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

Instructions for Minimum Change Migration

To configure the CIC Compiler v4.0 to most closely mimic the behavior of v2.0 the translation is:

XCO Parameters - Set HAS_DOUT_TREADY to 0. This makes the interface equivalent to v2.0 with ND enabled:

`s_axis_data_tvalid` is equivalent to ND.

`s_axis_data_tready` is equivalent to RFD.

`m_axis_data_tvalid` is equivalent to RDY.

If ND was previously disabled (that is, the ND XCO parameter was FALSE) then tie `s_axis_data_tvalid` to 1. This means data can always be supplied, and the data transfer is solely controlled by `s_axis_data_tready` (previously RFD).

If SCLR was previously set to TRUE then remember that the reset pulse is now active-Low and must be a minimum of two clock cycles long.

Special Considerations when Migrating to AXI

The conversion to AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx IP LogiCORE solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn`, and event signals, all inputs and outputs to the CIC Compiler core are conveyed using AXI4-Stream channels. A channel always consists of TVALID and TDATA plus additional ports (such as TREADY, TUSER and TLAST) when required and optional fields. Together, TVALID and TREADY perform a handshake to transfer a message, where the payload is TDATA, TUSER and TLAST.

For further details on AXI4-Stream interfaces, see the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 7].

Basic Handshake

Figure A-1 shows the transfer of data in an AXI4-Stream channel. TVALID is driven by the source (master) side of the channel and TREADY is driven by the receiver (slave). TVALID indicates that the values in the payload fields (TDATA, TUSER and TLAST) are valid. TREADY indicates that the slave is ready to receive data. When both TVALID and TREADY are TRUE in a cycle, a transfer occurs. The master and slave set TVALID and TREADY respectively for the next transfer appropriately.

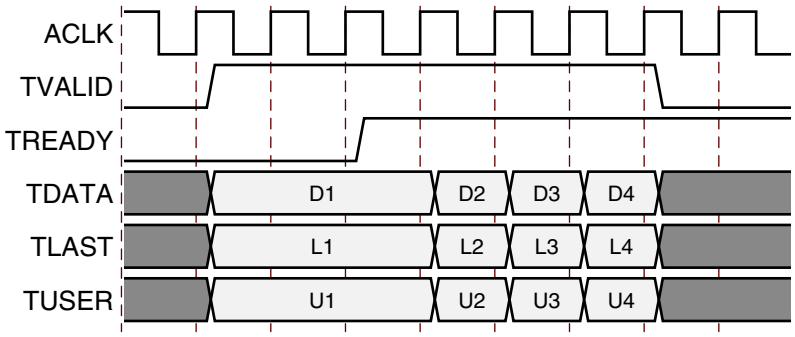

Figure A-1: **Data Transfer in an AXI4-Stream Channel**

AXI4 Channel Rules

All of the AXI4 channels follow the same rules:

- All TDATA and TUSER fields are packed in little endian format. That is, bit 0 of a sub-field is aligned to the same side as bit 0 of TDATA or TUSER.
- All TDATA and TUSER vectors are multiples of 8 bits. After all fields in a TDATA or TUSER vector have been concatenated, the overall vector is padded to bring it up to an 8 bit boundary.

Configuration Channel

This channel is only present in programmable rate configurations.

Pinout

The configuration channel pinout is shown in Table A-3.

Table A-3: **Configuration Channel Pinout**

Port Name	Port Width	Direction	Description
s_axis_config_tdata	Variable(1)	In	Carries the configuration information: RATE
s_axis_config_tvalid	1	In	Asserted by the external master to signal that it is able to provide data.
s_axis_config_tready	1	Out	Asserted by the CIC Compiler to signal that it is able to accept data.

Notes:
1. Range can be viewed in the Vivado IDE.

TDATA Fields

The configuration channel (s_axis_config) is an AXI4 channel that carries the fields shown in Table A-4 in its TDATA vector.

Table A-4: **Configuration Channel TDATA Fields**

Field Name	Width	Padded	Description
RATE	log2(maximum rate + 1)	Yes	The interpolation or decimation rate of the core.

The RATE field should be extended to the next 8 bit boundary if it does not already finish on an 8 bit boundary. The CIC Compiler core ignores the value of the padding bits, so they can be driven to any value. Connecting them to constant values can help reduce device resource usage.

TDATA Format

The configuration channel has one field which is packed into the s_axis_config_tdata vector as shown in Figure A-2 (starting from the LSB):

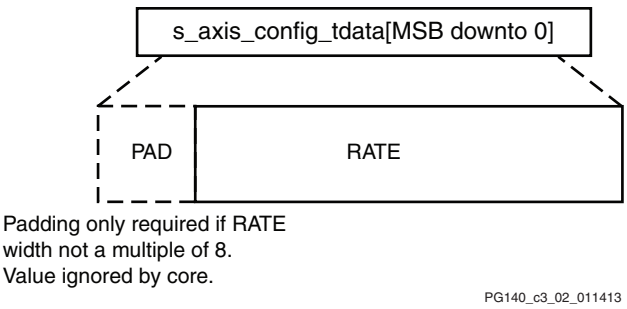

Figure A-2: **Configuration Channel TDATA (s_axis_config_tdata) Format**

Data Input Channel

The data input channel contains the sample data to be transformed.

Pinout

The data input channel pinout is shown in Table A-5.

Table A-5: **Data Input Channel Pinout**

Port Name	Port Width	Direction	Description
s_axis_data_tdata	Variable(1)	In	Carries the sample data
s_axis_data_tvalid	1	In	Asserted by the upstream master to signal that it is able to provide data

Table A-5: **Data Input Channel Pinout *(Cont'd)***

Port Name	Port Width	Direction	Description
s_axis_data_tlast	1	In	Asserted by the upstream master on the sample corresponding to the last channel. This is not used by the CIC Compiler except to generate the events: • event_tlast_unexpected • event_tlast_missing Only available when the core is configured to have multiple channels.
s_axis_data_tready	1	Out	Used by the CIC Compiler to signal that it is ready to accept data

Notes:
1. Range can be viewed in the Vivado IDE.

TDATA Fields

The DATA field is packed into the `s_axis_data_tdata` vector as shown in Table A-6.

Table A-6: **Data Input Channel TDATA Fields**

Field Name	Width	Padded	Description
DATA	2 to 24	Yes	The sample data to be processed in twos complement format.

The DATA field should be extended to the next 8-bit boundary if it does not already finish on an 8-bit boundary. The CIC Compiler core ignores the value of the padding bits, so they can be driven to any value. Connecting them to constant values can help reduce device resource usage.

TDATA Format

The data input channel TDATA vector (`s_axis_data_tdata`) has one field (DATA) which is packed as shown in Figure A-3.

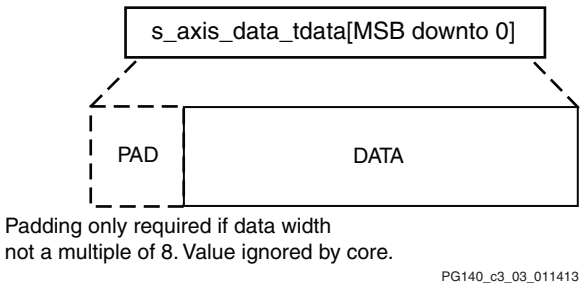

Figure A-3: **Data Input Channel TDATA (s_axis_data_tdata) Format**

Data Output Channel

The data output channel contains the processed samples, which are carried on TDATA. In addition, TUSER carries per-sample status information relating to the sample data on TDATA. This status information is intended for use by downstream slaves that directly process data samples. It cannot get out of synchronisation with the data as it is transferred in the same channel. This information is classed as per-sample status:

1. CHAN_SYNC
2. CHAN_OUT

Pinout

The data output channel pinout is shown in Table A-7.

Table A-7: **Data Output Channel Pinout**

Port Name	Port Width	Direction	Description
m_axis_data_tdata	Variable(1)	Out	Carries the sample data.
m_axis_data_tuser	Variable(1)	Out	Carries additional per-sample. Only available when the core is configured to have multiple channels.
m_axis_data_tvalid	1	Out	Asserted by the CIC Compiler to signal that it is able to provide sample data.
m_axis_data_tlast	1	Out	Asserted by the CIC Compiler on the sample corresponding to the last channel. Only available when the core is configured to have multiple channels.
m_axis_data_tready	1	In	Asserted by the external slave to signal that it is ready to accept data. Only available when the core is configured to have a TREADY on the data output channel.

Notes:
1. Range can be viewed in the Vivado IDE.

TDATA Fields

The DATA field is packed into the `m_axis_data_tdata` vector as shown in Table A-8.

Table A-8: **Data Output Channel TDATA Fields**

Field Name	Width	Padded	Description
DATA	2 to 104	Yes - sign extended	The processed sample data in twos complement format.

The DATA field is sign extended to the next 8-bit boundary if it does not already finish on an 8-bit boundary.

TDATA Format

The data output channel TDATA vector (`m_axis_data_tdata`) has one field (DATA) which is packed as shown in Figure A-4.

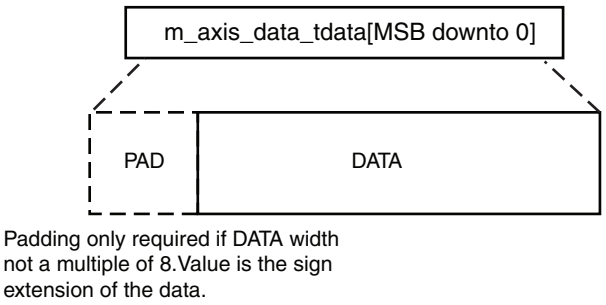
m_axis_data_tdata[MSB downto 0]
PAD
DATA
Padding only required if DATA width not a multiple of 8.Value is the sign extension of the data.

Figure A-4: **Data Output Channel TDATA (m_axis_data_tdata) Format**

TUSER Fields

The data output channel carries the fields shown in Table A-9 in its TUSER vector.

Table A-9: **Data Output Channel TUSER Fields**

Field Name	Width	Padded	Description
CHAN_OUT	log2 (number of channels)	Yes - zero-extended	The number of the channel (0 indexed) corresponding to the sample on m_axis_data_tdata.
CHAN_SYNC	1	Yes - zero-extended	Asserted with the sample corresponding to the first channel. Only available when the core is configured to have multiple channels.

All fields with padding should be zero-extended to the next 8-bit boundary if they do not already finish on an 8-bit boundary.

TUSER Format

The fields are packed into the `m_axis_data_tuser` vector in the order shown in Figure A-5. (starting from the LSB):

1. CHAN_OUT plus padding
2. CHAN_SYNC plus padding

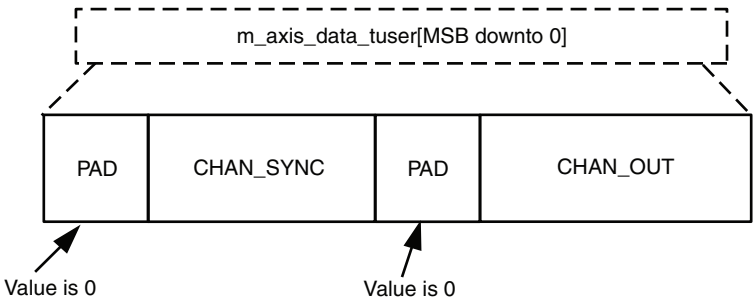

Figure A-5: **Data Output Channel TUSER (m_axis_data_tuser) Format**

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the CIC Compiler core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the CIC Compiler core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Records for the CIC Compiler

AR: 29297

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

Interface Debug

If data is not being transmitted or received, check the following conditions:

- If transmit `<interface_name>_tready` is stuck Low following the `<interface_name>_tvalid` input being asserted, the core cannot send data.
- If the receive `<interface_name>_tvalid` is stuck Low, the core is not receiving data.
- Check that the `ACLK` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed. See Protocol Description in Chapter 3.
- Check the core configuration.

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide.

1. Eugene B. Hogenauer, *An Economical Class of Digital Filters for Decimation and Interpolation*, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-29, No. 2, April 1981.
2. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
3. *Vivado Design Suite User Guide: Designing with IP* (UG896)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *ISE to Vivado Design Suite Migration Guide* (UG911)
7. *Vivado Design Suite: AXI Reference Guide* (UG1037)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	4.0	Added support for UltraScale+ families.
09/30/2015	4.0	• Corrected signal direction in Product Specification chapter. • Corrected range information in Design Flow Steps chapter.
04/02/2014	4.0	• Added link to resource utilization information.
12/18/2013	4.0	• Added UltraScale™ architecture support information. • Added Simulation, Synthesis, and Test Bench chapters. • Updated Migrating appendix.
10/02/2013	4.0	Minor updates to IP Facts table and Migrating appendix. Document version number advanced to match the core version number.
03/20/2013	1.0	Initial release as a product guide. This document derived from the *LogiCORE CIC Compiler Data Sheet* (DS845).

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

© Copyright 2012-2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.